Detailed Documentation on Multi-Stage Pipeline YAML Configuration

Introduction

This document provides a detailed explanation of a multi-stage pipeline YAML configuration in Azure Pipelines. The pipeline is designed to handle a complete CI/CD process, including compiling code, running security scans with Trivy, performing code quality analysis with SonarQube, building a Docker image, scanning the image with Trivy, and finally deploying the application to a Kubernetes cluster. Each stage is configured to run on a specific agent pool, and the pipeline is set up to execute these stages sequentially.

```yaml
trigger:
- none

pool:
  name: Aditya
  demands: agent.name -equals agent-1

stages:
- stage: Compile
  displayName: 'Compile Stage'
  jobs:
  - job: CompileJob
    displayName: 'Compile Job'
    pool:
      name: Aditya
      demands: agent.name -equals agent-1
    steps:
    - script: mvn compile
      displayName: 'Compile-Step'

- stage: Trivy_FS_Scan
  displayName: 'Trivy FS Stage'
  jobs:
  - job: TrivyFSJob
    displayName: 'TrivyFS Job'
    pool:
      name: Aditya
      demands: agent.name -equals agent-1
    steps:
    - script: trivy fs --format table -o trivy-fs-report.html .
      displayName: 'TrivyFS-Scan-Step'

- stage: SonarQube_Scan
  displayName: 'SonarQube Stage'
  jobs:
  - job: SonarQubeJob
    displayName: 'SonarQube Job'
    pool:
     name: Aditya
     demands: agent.name -equals agent-1
    steps:
    - task: SonarQubePrepare@5
      inputs:
        SonarQube: 'sonar-svc'
        scannerMode: 'CLI'
        configMode: 'manual'
        cliProjectKey: 'screte-santa'
        cliProjectName: 'screte-santa'
        cliSources: '.'
        extraProperties: |
          # Additional properties that will be passed to the scanner,
          # Put one key=value per line, example:
          # sonar.exclusions=**/*.bin
          sonar.java.binaries=.
    - task: SonarQubeAnalyze@5
      inputs:
        jdkversion: 'JAVA_HOME'

- stage: Build
  displayName: 'Build Stage'
  jobs:
  - job: BuildJob
    displayName: 'Build Job'
    pool:
      name: Aditya
      demands: agent.name -equals agent-1
    steps:
    - script: mvn package
      displayName: 'Build-Package-Step'

- stage: Docker
  displayName: 'Docker Stage'
  jobs:
  - job: DockerJob
    displayName: 'Docker Job'
    pool:
      name: Aditya
      demands: agent.name -equals agent-1
    steps:
    - script: mvn package
      displayName: 'Build-Package-Step'
    - task: Docker@2
      inputs:
        containerRegistry: 'docker-svc'
        repository: 'adijaiswal/santa'
        command: 'buildAndPush'
        Dockerfile: '**/Dockerfile'
        tags: 'latest'
```

```yaml
- stage: Trivy_Image_Scan
  displayName: 'Trivy Image Stage'
  jobs:
  - job: TrivyImageJob
    displayName: 'Trivy Image Job'
    pool:
      name: Aditya
      demands: agent.name -equals agent-1
    steps:
    - script: trivy image --format table -o trivy-fs-report.html adijaiswal/santa:latest
      displayName: 'Trivy-Image-Scan-Step'

- stage: K8_Deploy
  displayName: 'K8_Deploy Stage'
  jobs:
  - job: TK8_DeployJob
    displayName: 'K8_Deploy Job'
    pool:
      name: Aditya
      demands: agent.name -equals agent-1
    steps:
    - task: KubectlInstaller@0
      inputs:
        kubectlVersion: 'latest'
    - task: Kubernetes@1
      inputs:
        connectionType: 'Kubernetes Service Connection'
        kubernetesServiceEndpoint: 'k8-service-connection'
        namespace: 'default'
        command: 'apply'
        useConfigurationFile: true
        configuration: 'k8-dep-svc.yml'
        secretType: 'dockerRegistry'
        containerRegistryType: 'Container Registry'
        dockerRegistryEndpoint: 'docker-svc'
        forceUpdate: false
```

1. Pipeline Overview

This pipeline is designed to automate the following tasks:

1. **Compile the code** using Maven.
2. **Run a security scan** on the file system using Trivy.
3. **Perform code quality analysis** using SonarQube.
4. **Build the application** and package it using Maven.
5. **Create a Docker image** of the application and push it to a Docker registry.

6. **Run a security scan** on the Docker image using Trivy.
7. **Deploy the application** to a Kubernetes cluster using kubectl.

Each task is organized into stages, and the pipeline is designed to ensure that all tasks are completed in the correct order, with appropriate checks and balances at each step.

2. Pipeline Trigger Configuration

yaml

Copy code

trigger:

- none

Explanation:

- The pipeline is configured with trigger: none, meaning it will not automatically trigger on any branch or path changes. This setup is typically used when the pipeline is intended to be triggered manually or from another pipeline.

Use Cases:

- This configuration is useful for pipelines that are part of a more extensive CI/CD process and are triggered by other pipelines or schedules rather than code commits.

3. Agent Pool Configuration

```
pool:
  name: Aditya
  demands: agent.name -equals agent-1
```

Explanation:

- The pipeline uses an agent pool named Aditya.
- The demands clause specifies that the agent must be named agent-1.
- This setup ensures that all jobs within the pipeline are executed on a specific agent, which may be necessary due to environment constraints or dependencies installed on the agent.

Use Cases:

- Using specific agents is essential when certain tasks require a unique environment, such as specific tools or configurations that are not available on other agents.

4. Stages and Jobs

The pipeline is divided into seven distinct stages, each with a specific purpose. Each stage contains one or more jobs, and each job consists of several steps that execute the required tasks.

1. **Compile Stage**: Compiles the source code.
2. **Trivy File System Scan Stage**: Scans the file system for vulnerabilities using Trivy.
3. **SonarQube Scan Stage**: Analyzes the code for quality and security using SonarQube.
4. **Build Stage**: Builds and packages the application.
5. **Docker Stage**: Builds a Docker image and pushes it to a registry.
6. **Trivy Image Scan Stage**: Scans the Docker image for vulnerabilities using Trivy.
7. **Kubernetes Deployment Stage**: Deploys the application to a Kubernetes cluster.

5. Detailed Breakdown of Each Stage

Compile Stage

```
- stage: Compile
  displayName: 'Compile Stage'
  jobs:
  - job: CompileJob
    displayName: 'Compile Job'
    pool:
      name: Aditya
      demands: agent.name -equals agent-1
    steps:
    - script: mvn compile
      displayName: 'Compile-Step'
```

Explanation:

- **Stage Name**: Compile
- **Job Name**: CompileJob
- **Task**: The job runs the mvn compile command to compile the source code.
- **Agent Pool**: The job runs on an agent from the Aditya pool, specifically agent-1.

Key Points:

- Compilation is the first step in the pipeline, ensuring that the source code is error-free and ready for further processing.
- The use of a dedicated agent ensures a consistent environment for the compilation process.

Trivy File System Scan Stage

```
- stage: Trivy_FS_Scan
  displayName: 'Trivy FS Stage'
  jobs:
  - job: TrivyFSJob
    displayName: 'TrivyFS Job'
    pool:
      name: Aditya
      demands: agent.name -equals agent-1
    steps:
    - script: trivy fs --format table -o trivy-fs-report.html .
      displayName: 'TrivyFS-Scan-Step'
```

Explanation:

- **Stage Name**: Trivy_FS_Scan
- **Job Name**: TrivyFSJob
- **Task**: The job runs a Trivy scan on the file system, outputting the results to trivy-fs-report.html.
- **Agent Pool**: The job runs on an agent from the Aditya pool, specifically agent-1.

Key Points:

- Trivy scans the entire file system for vulnerabilities, ensuring that any known security issues are detected early in the process.
- The scan results are saved in an HTML file, which can be reviewed to address any vulnerabilities.

SonarQube Scan Stage

```
- stage: SonarQube_Scan
  displayName: 'SonarQube Stage'
  jobs:
  - job: SonarQubeJob
    displayName: 'SonarQube Job'
    pool:
      name: Aditya
      demands: agent.name -equals agent-1
    steps:
    - task: SonarQubePrepare@5
      inputs:
        SonarQube: 'sonar-svc'
        scannerMode: 'CLI'
        configMode: 'manual'
        cliProjectKey: 'screte-santa'
        cliProjectName: 'screte-santa'
        cliSources: '.'
        extraProperties: |
          sonar.java.binaries=.
    - task: SonarQubeAnalyze@5
```

```
      inputs:
        jdkversion: 'JAVA_HOME'
```

Explanation:

- **Stage Name**: SonarQube_Scan
- **Job Name**: SonarQubeJob
- **Task**: The job prepares and runs a SonarQube analysis on the source code.
- **Agent Pool**: The job runs on an agent from the Aditya pool, specifically agent-1.

Key Points:

- SonarQube is used to analyze the source code for quality issues, security vulnerabilities, and coding standards.
- The results help maintain high code quality and adhere to best practices.

Build Stage

```
- stage: Build
  displayName: 'Build Stage'
  jobs:
  - job: BuildJob
    displayName: 'Build Job'
    pool:
      name: Aditya
      demands: agent.name -equals agent-1
    steps:
    - script: mvn package
      displayName: 'Build-Package-Step'
```

Explanation:

- **Stage Name**: Build
- **Job Name**: BuildJob
- **Task**: The job runs the mvn package command to build and package the application.
- **Agent Pool**: The job runs on an agent from the Aditya pool, specifically agent-1.

Key Points:

- The build stage compiles the application into a deployable artifact, typically a JAR or WAR file for Java projects.
- The packaging process ensures that the application is ready for deployment.

Docker Stage

```
- stage: Docker
  displayName: 'Docker Stage'
  jobs:
  - job: DockerJob
    displayName: 'Docker Job'
    pool:
      name: Aditya
      demands: agent.name -equals agent-1
    steps:
    - script: mvn package
      displayName: 'Build-Package-Step'
    - task: Docker@2
      inputs:
        containerRegistry: 'docker-svc'
        repository: 'adijaiswal/santa'
        command: 'buildAndPush'
        Dockerfile: '**/Dockerfile'
        tags: 'latest'
```

Explanation:

- **Stage Name**: Docker
- **Job Name**: DockerJob
- **Tasks**:
  1. The job rebuilds the application using mvn package.
  2. The Docker task builds a Docker image from the Dockerfile and pushes it to the specified container registry.
- **Agent Pool**: The job runs on an agent from the Aditya pool, specifically agent-1.

Key Points:

- Docker images are built and pushed to a registry, making them available for deployment.
- The image is tagged with latest, indicating it as the most recent build.

Trivy Image Scan Stage

```
- stage: Trivy_Image_Scan
  displayName: 'Trivy Image Stage'
  jobs:
  - job: TrivyImageJob
    displayName: 'Trivy Image Job'
    pool:
      name: Aditya
      demands: agent.name -equals agent-1
    steps:
    - script: trivy image --format table -o trivy-fs-report.html adijaiswal/santa:latest
      displayName: 'Trivy-Image-Scan-Step'
```

Explanation:

- **Stage Name**: Trivy_Image_Scan
- **Job Name**: TrivyImageJob
- **Task**: The job runs a Trivy scan on the Docker image that was built and pushed in the previous stage. The scan results are output in a table format to trivy-fs-report.html.
- **Agent Pool**: The job runs on an agent from the Aditya pool, specifically agent-1.

Key Points:

- Trivy scans the Docker image for known vulnerabilities, ensuring that the image is secure before being deployed to production.
- The scan results provide a detailed report of any vulnerabilities found, allowing teams to address them before deployment.

Kubernetes Deployment Stage

```
- stage: K8_Deploy
  displayName: 'K8_Deploy Stage'
  jobs:
  - job: K8_DeployJob
    displayName: 'K8_Deploy Job'
    pool:
      name: Aditya
      demands: agent.name -equals agent-1
    steps:
    - task: KubectlInstaller@0
      inputs:
        kubectlVersion: 'latest'
    - task: Kubernetes@1
      inputs:
        connectionType: 'Kubernetes Service Connection'
        kubernetesServiceEndpoint: 'k8-service-connection'
        namespace: 'default'
        command: 'apply'
        useConfigurationFile: true
        configuration: 'k8-dep-svc.yml'
        secretType: 'dockerRegistry'
        containerRegistryType: 'Container Registry'
        dockerRegistryEndpoint: 'docker-svc'
        forceUpdate: false
```

Explanation:

- **Stage Name**: K8_Deploy
- **Job Name**: K8_DeployJob
- **Tasks**:
    1. The job installs the kubectl command-line tool, ensuring the correct version is available for deployment.

    2. The Kubernetes task deploys the application to a Kubernetes cluster using the specified k8-dep-svc.yml configuration file.
    3. The deployment is performed using a Kubernetes service connection configured in Azure DevOps.
- **Agent Pool**: The job runs on an agent from the Aditya pool, specifically agent-1.

Key Points:

- The Kubernetes deployment stage automates the process of deploying the Docker image to a Kubernetes cluster.
- The kubectl apply command applies the deployment configuration, creating or updating Kubernetes resources based on the provided YAML file.
- The use of a Docker registry secret ensures that the Kubernetes cluster can securely pull the Docker image from the specified registry.

6. Best Practices

1. **Agent Pool Consistency**:
    - Ensure that the specified agent pool (Aditya) and agent (agent-1) are consistently used across all stages to maintain environment consistency.
2. **Security Scans**:
    - Regularly update Trivy and SonarQube to the latest versions to ensure that the most recent vulnerabilities are detected.
    - Review the security scan reports (Trivy and SonarQube) after each pipeline run and address any identified issues promptly.
3. **Pipeline Triggers**:
    - Consider enabling triggers based on branch changes or pull requests to automate the pipeline execution when new code is merged into the main branch.
4. **Kubernetes Deployment**:
    - Use a dedicated namespace for each environment (e.g., development, staging, production) to avoid conflicts and maintain a clear separation between deployments.
    - Implement rollback mechanisms in case of deployment failures, such as using Kubernetes' built-in deployment strategies (e.g., rolling updates).
5. **Artifact Management**:
    - Store build artifacts and Docker images in a secure, versioned repository to ensure traceability and reproducibility of deployments.

7. Conclusion

This multi-stage pipeline YAML configuration provides a comprehensive CI/CD process, covering everything from compiling code to deploying it to a Kubernetes cluster. By integrating security scans with Trivy and SonarQube, the pipeline ensures that both the source code and the Docker images are free from known vulnerabilities. The use of Docker and Kubernetes allows for scalable, containerized deployments, while the consistent use of an agent pool ensures that the pipeline runs in a controlled environment.

This setup can be adapted and expanded based on specific project requirements, such as adding more stages for testing, using different deployment strategies, or integrating with additional tools like Helm or Terraform for infrastructure management.